matRiks: An R package for the automatic generation of rule-based matrices

by Quietest Quokka and Bounciest Bilby

Abstract Few resources are available for the automatic generation of Raven-like matrices. Some of them are no longer working, while others are hardly customizable without strong programming skills. An R package fro the automatic generation of stimuli used for psychological assessment exists, but it is limited to one shape and to the manipulation of one rule (i.e., rotation). The **matRiks** package has been developed with the aim of overcoming the above mentioned issues. This package can generate matrices considering different types of rules, starting from the most basic ones (e.g., changes in size, objects orientation) to the most complex ones, based on inferential and inductive reasoning. This unveils the possibility of generating new customizable stimuli and of systematically manipulating the difficulty of the matrices. Being developed within the R environment, the **matRiks** package is completely open-source, allows for the reproducibility of the stimuli, and it can be easily used by people with basic knowledge of R language.

1 Introduction

Cattell (Cattell (1963)) defined fluid intelligence (g) as the ability of solving novel reasoning problems that has little to do with concepts learned in schools or through acculturational processes. The adjective "fluid" explicitly refers to its ability to "flow" into a variety of tasks and cognitive activities (Horn, 1972). Given this definition of fluid intelligence, it appears natural that the instruments for used for its evaluation tap on the respondent's ability to solve abstract problems that involve acculturation as little as possible, such as figural analogies, figure classifications, matrices, and number and letter series are often used (Horn, 1968).

The Raven's progressive matrices (RPM, JC ea Raven (1938)) are among the most famous tools for the assessment of g. The RPM consists in a series of non-verbal multiple-choice stimuli where respondents are required to complete a series of drawings composed of different figures by identifying the relevant features that rule the relationships between the figures. These drawings are often referred to as matrices. To pursue this aim, the respondents must choose the figure that complete the drawing among a list of other figures, the so-called distractors. This task should measure the ability of the respondents to identify and take into account the features (also called "rules") that govern the relationship between the figures to compose the drawing. The RPM and similar tasks (here denoted as Raven-like matrices or Raven-like tasks) are employed in different fields, from clinical evaluation of intelligence to the selection processes in organizational psychology (citation needed). Since Raven's and Raven-like tasks involve the ability to solve new abstract problems, the stimuli composing these tasks should not be spread among the general population. However, new stimuli can be generated according to the rules that govern the relationships between the figures composing the drawing. Indeed, different resources are available for developing Raven-like tasks, such as Sandia (cit), Corvus (cit), and the R package **Imak** (cit).

The stimuli generated with Sandia have been analysed in an Item Response Theory framework to validate them as a test for measuring fluid intelligence. The stimuli are available upon request to the authors, however no new stimuli can be generated because the code on which Sandia is based is no longer maintained. Corvus represents another possible resource for generating Raven-like tasks. Corvus is written in Javascript but the Author provided a nice and easy-to-use graphical interface where the user can specify the figures and the rule(s) for the generation of the matrices. However, Corvus provides few degrees of freedom in terms of both the figures and the number of rules that can be manipulated through the graphical interface. If the user wants to add other figures, to modify the already existing figures, or to implement new rules, they have to modify the source code in Javascript, which might be a quite demanding task for people with little to null experience in programming. Finally, the `ImaK` package is an `R` package that allows for generating visual analogies. The code for generating such stimuli (along with their response options) is quite straightforward and easy to use. However, the stimuli that can be generated with the `ImaK` package are mostly based on the rotation of the same figure to which some objects can be added or removed. As such, the only rule that is manipulated is the spatial rotation of the figures.

Given the limitations of the existing resources for generating Raven-like tasks, there might be the need of an open-source, easy-to-use, and constantly maintained resource for generating such stimuli through the systematic manipulation of rules applied to different figures. The `matRiks` package (Brancaccio, Epifania, and de Chiusole (2023)) has been developed to pursue these aims. Beyond

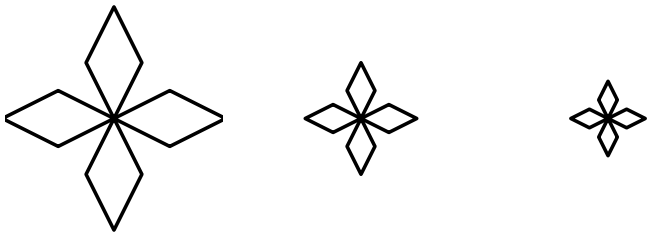


Figure 1: Example of visuospatial rule: Changes in size

generating the matrix by manipulating one or multiple rules at once on one or multiple figures, the `matRiks` package generates the response list associated to the matrix as well. The systematic manipulation of both the rules and the figures for the matrix generation should grant the possibility of grading the granularity of the difficulty of the matrices by manipulating one element at the time. In a similar vein, the package should allow for generating matrices that can be considered equivalent in terms of rules employed for their generation but differ in terms of figures composing the drawing. In what follows, the term stimulus is used to identify the matrix with its associated response list.

The manuscript is organized as follows. The next section presents the rules that usually employed in the RPM along with the specific types of error responses (i.e., distractors) that compose the response list associated with a matrix. Then, the `matRiks` package is presented through an example of its application for the generation of different stimuli (i.e., the matrix with its associated response list).

2 Background

Rule based matrices

Literature highlights a plethora of rules that can be manipulated for the generation of the raven-like tasks (cit cit cit). Beyond the fact that some of these rules have different names in different sources but refer to the same manipulation (e.g., the rule defined as "and problem" in Harris el a. 2020 is called "intersection" rule in Arendasy et al. 2005), they can be summarized into different macro-categories, namely visuospatial rules (i.e., the manipulation concerns the graphical and spatial features of the figures, Figure 1), and logical rules (i.e., the manipulation concerns the logical relationships between the figures composing the matrix, , Figure 2). The rules can be applied (i.e., manipulated) to different figures or concatenation of figures to generate a matrix.

In Figure 1, the manipulation concerns a specif feature of the figure, that is its size, and it can be observed as the the figure decreases its size across the cells. The rightmost cell contains the figure with the smallest size, the middle cell contains a figure with medium-small size while the leftmost object contains the figure with its original size. In Figure 2, the manipulation concerns the relationships between the objects composing the figures, which are combined together according to a logical rule based on the insiemistic intersection of the objects. Specifically, the figure in the rightmost cell results from the intersection of the objects in the leftmost cell and in the middle cell.

Both visuospatial and logical rules can be manipulated according to different directional logic. Specifically, the rules can be applied horizontally (i.e., the manipulation of the rule can be seen across columns but not across rows, H direction), vertically (i.e., the manipulation of the rule can be seen across rows but not across columns, V direction), or diagonally (i.e., the manipulation of the rule can be seen both across columns and across rows). Concerning the diagonal directional logic, it can follow either the main diagonal of the matrix (i.e., the manipulation of the rule can be seen from the top-left corner to the low-right corner, TL-LR direction) or the secondary diagonal of the matrix (i.e., the manipulation of the rule can be seen from the low-left corner to the top-right corner, LL-TR direction).

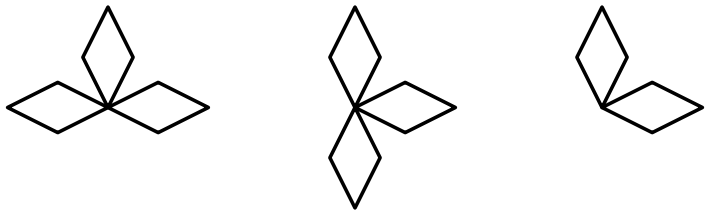


Figure 2: Example of logical rule: Insiemistic Interscetion AND

The response options

A large corpus of literature has investigated the role of the distractors in the response processes involved when solving the Raven matrices, focusing on the specific error response chosen by the respondent (Forthmann et al. (2020), Kunda et al. (2016), Storme et al. (2019)). The underlying logic is that the incorrect response is not chosen at random by the respondent, but it can be the result of an educated guess, or it can be chosen because the respondent is misled for a definite reason. In other words, the incorrect responses might reflect an incorrect solution strategy which results in the choice of a specific distractor type over another one (Kunda et al. 2016). The distractors can be classified according to the incorrect response strategy they represent. Kunda et al. (2016) present a list of criteria for the identification of the distractors in the SPM based on the error types from the CPM and APM manuals (John Raven and Raven 2004). Specifically, the specific response that is chosen in place of the correct one (i.e. error types) can be collected into four main four conceptual errors, namely Repetition (R), Difference (D), Wrong Principle (WP), and Incomplete Correlate (IC). Repetition errors occur when the chosen response option is a cell adjacent to the blank space. Difference errors occur when the chosen response option is completely different from any entry of the matrix. Wrong principle errors occur when the chosen response option follows rules other than the ones used in the matrix. Incomplete Correlate errors occur when the chosen response option is in fact the correct response with a variation on a single feature. Each of the four main error types can be further described by their subcategories.
Table 1 illustrates the definitions of the error types.

The criteria for the classification of the error types were used for the formal definition and generation of the distractors implemented in the `matRiks` package. These criteria were included in the response options operator with the aim of providing the user with a response list composed of 11 elements (ten distractors and the correct response) among which they could choose the most appropriate ones.

The response options operator generates a response list composed of the correct response, three reptition distractors, one difference distractor, two wrong principle distractors, and four incomplete correlate distractors. Further details on the formal definition of each of the distractors and on their generation are given in the "Generation of the response list" Section.

3 The matRiks package

The `matRiks` (Brancaccio, Epifania, and de Chiusole 2023) package can generate 2×2 and 3×3 Raven-like matrices with their corresponding set of responses (i.e., the correct response and all the distractors described in the Generation of response list section). The Raven-like matrices can be generated according to either visuo-spatial or logic rules, which can be concatenated with three different directional logic, namely vertically, horizontally, and diagonally. Finally, it is possible to print the generated matrices and set of distractors as either single images (i.e., each cell of the matrix and each distractor are printed separately) or as a complete figure with the set of single distractors.

Table 1: Taxonomy of error types

Distractors	Definition
Repetition (R)	Entries of the matrix adjacent to the blank cell
Difference (D)	Combination of all the entries of the matrix or the combination of some of their fea
Wrong Principle (WP)	Copy or combination of the matrix entries according to another rule
Incomplete Correlate (IC)	Correct response with a variation on only a single feature.

Installation

The `matRiks` package is available on CRAN and can be installed as:

```
install.packages("matRiks")
```

The code `vignette(package ="matRiks")` allows for obtaining the list of all the vignettes included in the package. Each vignette can be accessed via `vignette("vignette-name", package="matRiks")`. For instance, the code `vignette("generate_matriks", package ="matRiks")` opens the vignette that contains the instruction on how to generate an `RMarkdown` file where both the matrix and its associated response options are plotted together.

Definition of figures

The `matRiks` package contains a high number of default figures that can be used for the generation of the matrices. All figures are defined as functions, hence their name has to be followed by the parentheses, and they have class `figure`. The arguments that can be modified inside the parentheses might vary from figure to figure, and they allow for changing different features of the figure. Specifically, the arguments that can be modified inside of the parentheses modify the default features of the figures, which are stored in a list of length 15. this list contains other lists with all the features of the figure. The features of the figure are as follows:

- `shape`: character, the name of the figure
- `size.x`: numeric, the length of the semi-major axis of the ellipse within which the figure is inscribed (see the documentation of the **DescTools** for further details)
- `size.y`: numeric, the length of the semi-minor axis of the ellipse within which the figure is inscribed (see the documentation of the **DescTools** for further details)
- `theta.1`: numeric, radiants of the rotation of the circle
- `theta.2`: numeric, radiants of the rotation of the circle
- `rotation`: numeric, radiants of the rotation of the ellipse within which the figure is inscribed
- `pos.x`: numeric, the position on the x-axis
- `pos.y`: numeric, the position on the y-axis
- `lty`: integer, the line type of the margins of the figure
- `lwd`: integer, the width of the margins of the figure
- `num`: don't remembed
- `nv`: integer, the numberof vertixes of the figures (might vary from 2 –lines– to 100 –circle and ellipse–)
- `shade`: character, the filling of a figure (can also be NA –empty figure–)
- `visible`: inetger, the visibility of the figure
- `tag`: character, properties of the figure used for the definition of the distractors

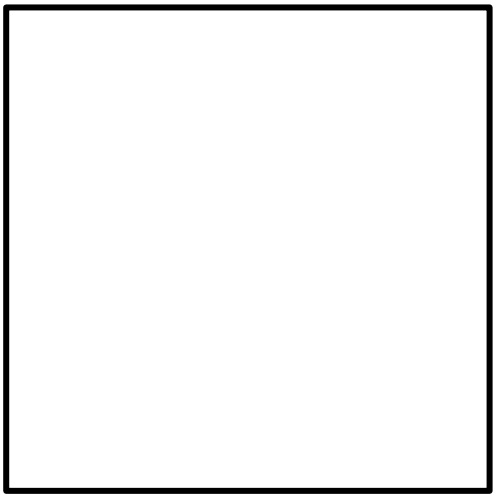


Figure 3: A simple square

For instance, a simple square (Figure @ref{fig:square}) can be printed with the command line:

```
draw(square())
```

The figures can be summarized in different categories, and for each of these categories there is a vignette that list all the figures included in that specific category.

The following types of figure are available in the matRiks package. For each type of figures, there is a vignette availbale that lists the entire figures for each type:

Concatenation of figures

Other then the pre-existing figures, the `matRiks` package allow for the generation of new figures by concatenating the existing ones. The `cof()` (**c**oncatenation **o**f **f**igures) function is designed for this aim. The arguments of the `cof()` function are the names of the default figures presented in the previous section.

For instance, the figure in Figure 11 is obtained by concatenating a `circle()` and a `dot()`.

```
eye <- cof(circle(), dot()) # create the new figure eye by concatenating the circle and the dot
draw(eye)
```

The resulting figure `eye` is a named list of lists of class `figure`. However, the two original figures are still available and can be considered as separated entities.

```
eye$shape
```

```
#> [1] "circle" "dot"
```

Thus, the object eye is considered a concatenation of two figures.

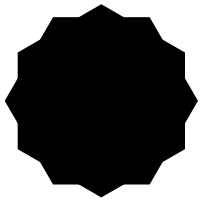


Figure 4: [Black figures](https://cran.r-project.org/web/packages/matRiks/vignettes/black-figures.html)

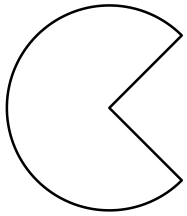


Figure 5: [Circle sections](https://cran.r-project.org/web/packages/matRiks/vignettes/circle-sections.html)

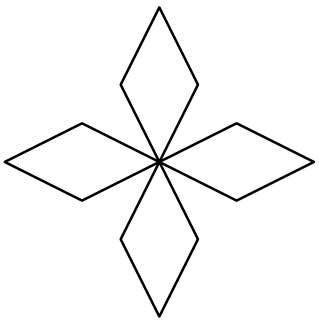


Figure 6: [Other figures](https://cran.r-project.org/web/packages/matRiks/vignettes/other-figures.html)

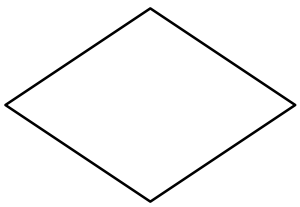


Figure 7: [Closed figures](https://cran.r-project.org/web/packages/matRiks/vignettes/closed-figures.html)

Figure 8: [Eight-shaped figures](https://cran.r-project.org/web/packages/matRiks/vignettes/eight-shapes-figures.html)

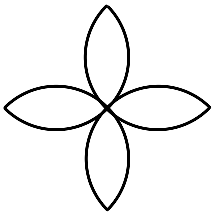


Figure 9: [Flowers figures](https://cran.r-project.org/web/packages/matRiks/vignettes/flowers-figures.html)

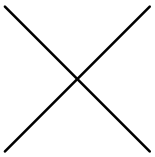


Figure 10: [Lines](https://cran.r-project.org/web/packages/matRiks/vignettes/lines.html)

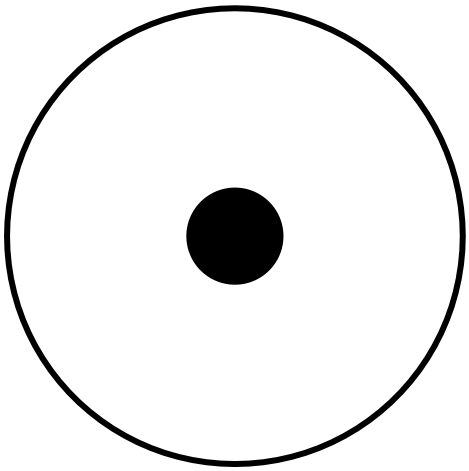


Figure 11: Example of concatenation of circle and dot to obtain an eye-like figure.

Table 2: Four-cell matrix

Sq1	Sq3
Sq2	Sq4

Function `cof()` has also two optional arguments `single` and `name`. If set to `TRUE` , the first argument forces the outcome of the concatenation to be consider as a new single. The second argument defines a new name for such a figure. The following code recreated the figure in Figure 11 but it forces the new figure to be a single figle named "eye".

```
s_eye <- cof(circle(),dot(),single = TRUE, name = "eye")
s_eye$shape

#> [1] "eye"
```

This difference will be relevant for the next sections in which there are rules that require a certain number of figures to be applied.

Available rules and matrix generation

The function `mat_apply(Sq1, hrules="identity", vrules ="idenity", mat.type=9)` is the main function for the generation of matrices based on rules and returns an object of class `matriks`. This function allows for the generation of matrices of different dimension, either 4-cell or 9-cell matrices. The dimension of the matrix can be specified with the argument `mat.type`, such that `mat.type = 4` results in 4-cell matrices (Table 2)

and `mat.type = 9` results in 9-cell matrices (default, Table Table 3).

The `Sq1` argument defines the starting figure (i.e., the figure to be plotted in the first cell `Sq1`), which can also be a concatenation of figures. The arguments `hrules` and `vrules` allow for the definition of the directional logic with the rule(s) is applied, such that the rules specified in `hrules` are manipulated horizontally and those specified in `vrules` are applied vertically.

Table 3: Nine-cell matrix

Sq1	Sq4	Sq7
Sq2	Sq5	Sq8
Sq3	Sq6	Sq9

The application of the `mat_apply()` function results in an object of class `matriks`, which is a named list. The length of the list vary as the dimension of the matrix varies, such that it has length 7 when `mat.type =4` and of length 12 otherwise. Regardless of the length of the list, it contains the characteristics of the matrix. the following example is based on a 9-cell matrix where the identity rule was applied both horizontally and vertically:

```
#>  [1] "Sq1"      "Sq2"      "Sq3"      "Sq4"      "Sq5"      "Sq6"
#>  [7] "Sq7"      "Sq8"      "Sq9"      "hrule"    "vrule"    "mat.type"
```

In particular, `Sq1` to `Sq9` are lists containing the figures for each of the nine cells composing the matrix, while `hrule`, `vrule`, `mat.type` are vectors containing the rules applied the rules applied horizontally, vertically, and the dimension of the matrix, respectively. Since this is a 9-cell matrix, the list is of length 12 and it contains the cells from Sq1 to Sq9. If a 4-cell matrix is generated, then the list if of length 7 and it contains the cells from Sq1 to Sq4, along with `hrule`, `vrule`, `mat.type` .

The rules are methods that transform a feature of the figure to obtain a different figure or a figure with a modified feature. The `matRiks` package implements several rules, each of which may be classified differently. The table contains all the rules available in the package, their descriptions, and classifications.

[Rule table]

As already mentioned, the rules can be classified into visuospatial and logical rules. Such macro classification may have an impact on the difficulty of the generated matrix [cita]. By the way in which the rules have been defined in the **matRiks** package, they can be either incremental or permutational rules.

The operation underlying the functioning of the incremental rules is the increment of a fixed quantity. Incremental rules apply a fixed increase (or decrease) on each cell of the matrix to obtain the feautures of the figure in the following cell, starting from the first cell. The order in which the fixed increase (decrease) is applied depends on the directional logic used for the generation of the matrix. For instance, Figure 12 illustrates the application of the change in size rule on a square. the change in size rule is an incremental rule, in which the size of the square decreases of a fixed quantity in each subsequent object according to a directional logic.

The incremental aspects appear because the squares are smaller compared to the ones on their right but larger than the ones on their left. Hence, by changing the position of any figure in the row the results would not necessarily respect the definition of the rule.

The operation underlying the functioning of the permutational rules is the permutation of the figures (or of their features) across the cells according to a directional logic. For instance, the change in shapes rule is a permutational rule where the objects shown in each cell of the matrix are permuted following an order that is consistent with the directional logic. For instance, consider a matrix where a set of three figures (`hexagon()`, `pentagon`, `square`) are manipulated according to the shape rule following an horizontal directional logic (Figure 13).

By default, the order of the figures is the order with which the figures are concatenated in the starting set. If the order of the figures in the original set is changed, then the rule will adapt and the result will be consistent with the new starting order.

All the rules functions are characterized by three arguments, namely `fig`,`n`, and `rules`. The `fig` argument defines the initial figure that will be transformed. The `n` argument defines the position of the resulting figure in the cell, in particular the number of rows or columns (depending on the directional logic) in which the outcome will be displayed. Finally, the `rules` argument is a vector of characters that defines the type of rule. In particular, if the vector `rules` contains the strings `inv` the default order is reversed, and it is denoted that the rule is applied reverse. For instance, to reverse the change in size in Figure **??**fig:incremental), the code `mat_apply(square(size.x=10), hrules = "size.inv")` should be used. Figure 14 depicts the resultin sequence of squares.

The reverse application of the rules can be done with permutational rules as well. By starting with the same set of figures as the one used to generate Figure 13, the sepcification of the argument `hrules = "shape.inv"` would result in a reverse ordering of the figures across the cells (Figure 15).

Even though rules are applied to a single figure their purpose is to be cohesive inside a matrix to allow the participant the correct options responses.

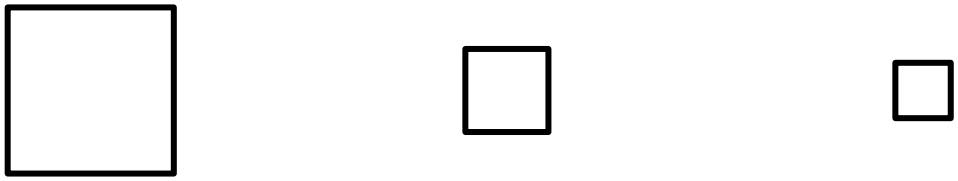


Figure 12: Example of size rule transformation along a row.

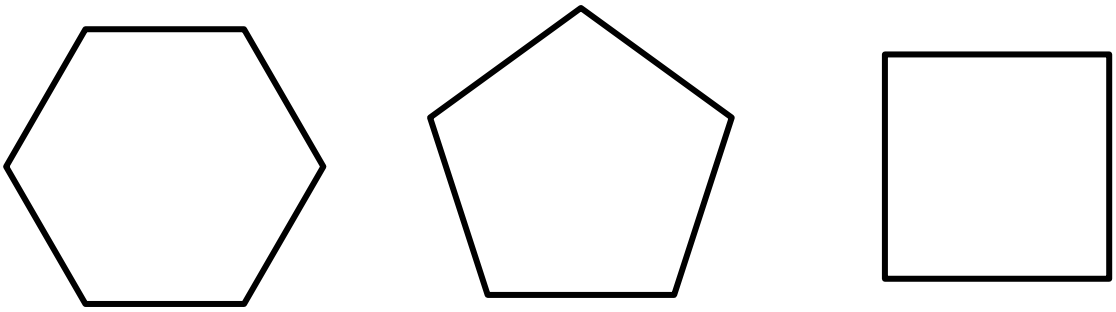


Figure 13: Example of shape rule transformation along a row.

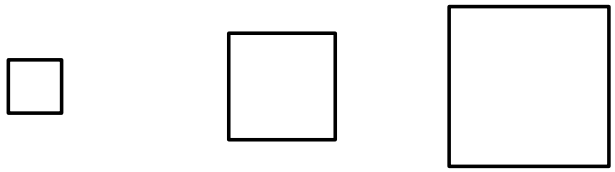


Figure 14: Example of incremental rule with reverse application: Change in size

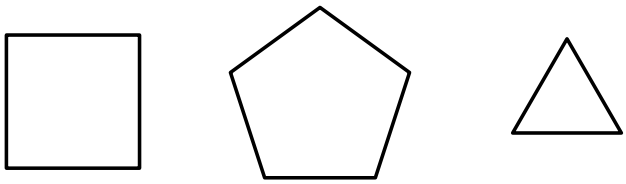


Figure 15: Example of permutational rule with reverse application: Change of shape

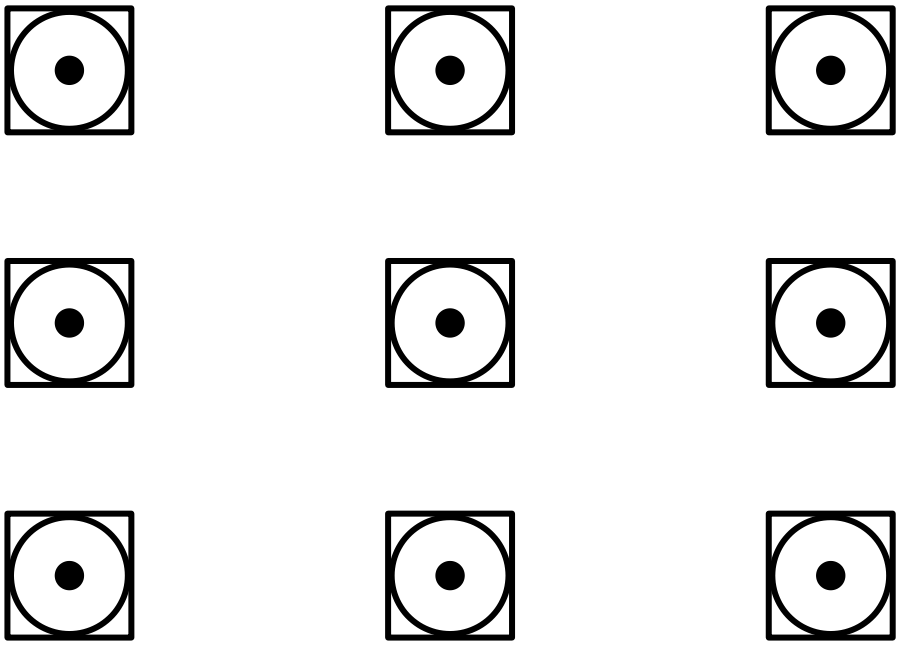


Figure 16: Example of outcome of the first step of the mat_apply procedure.

Logical rules `AND`, `OR` and `XOR` are a special case of incremental rules. Specifically, these rules cannot be applied in reverse.

Figure @ref(fig:??) shows the flowchart of the generative procedure underlying `mat_apply()`. The procedure consists of three serial steps. For the sake of simplicity, they will be illustrated for the 9-cell matrices only. A similar line of reasoning can be applied to 4-cell matrices.

The first step is to generate the named list of all the cells from `Sq1` to `Sq9`. Each of the cells contains the initial figure defined in `Sq1`. No rule has still been applied. For instance, considering the concatenation of figure `cof(square(), circle(),dot())` plotting the outcome of this step will result in the following matrix:

Since the application of the logical rules requires a well-defined relationship between the cells of the matrix, the second step checks whether arguments `hrules` or `vrules` contain a logical rule. If so, the procedure throws an error. An error is thrown whenever the application of a logical rule is combined with the application of a visuospatial rule, a logical rule is used for the generation of a 4-cell matrix, or if two different logical rules are applied concurrently to the same figures with different directional logic. If only one single logical rule is applied and none of the above error-condition has been met, the procedure will generate the whole matrix in a single step.

If the test of the second step turns out to be false, the procedure enters in the third step. In this step, the rules specified in `hrules` are applied following the horizontal directional logic (i.e., the rules are iterated among the elements horizontally.

The fourth step applies the rules specified in vrules following the vertical directional logic (i.e., the rules are iterated among the elements horizontally).

Since the third and forth steps are made one after another using the same rule, for instance `size`, both with a horizontal a vertical logic, the two transformation on the cells ends up to cumulate. For instance, considering the code `mat_apply(square(),hrules="size", vrules="size")`, at the end of the third step the figure in `Sq5` will be half the size of `Sq1`. During the forth the size of `Sq5` will be halve again resulting in a figure that is a quarter of the initial `Sq1`.

It is worth mentioning that the combination of the same rule horizontally and vertically ends up in the definition of a diagonal directional logic. In particular, if the same rule is applied in reverse in

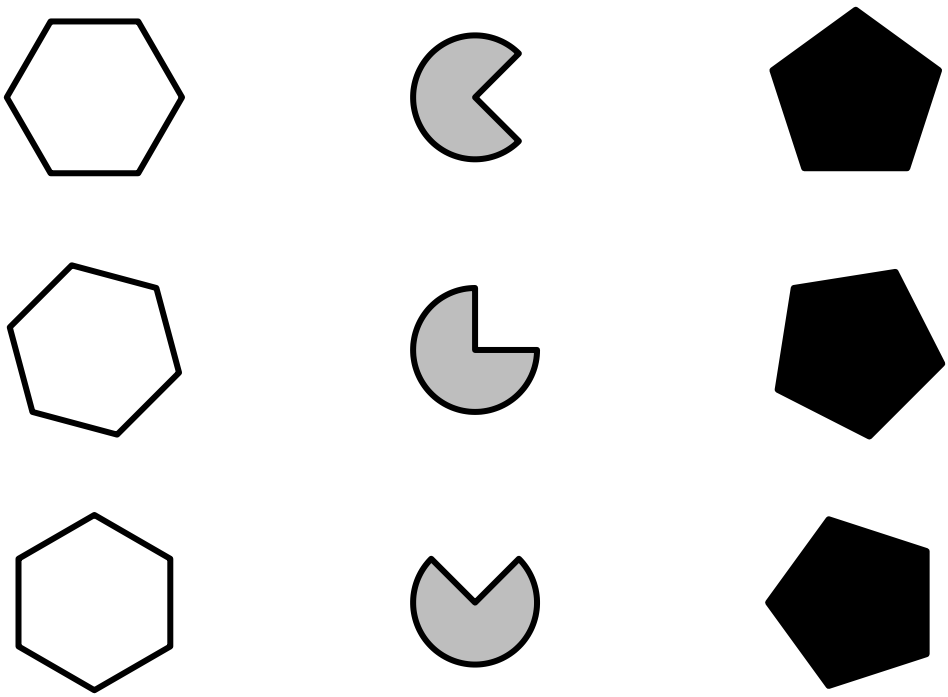


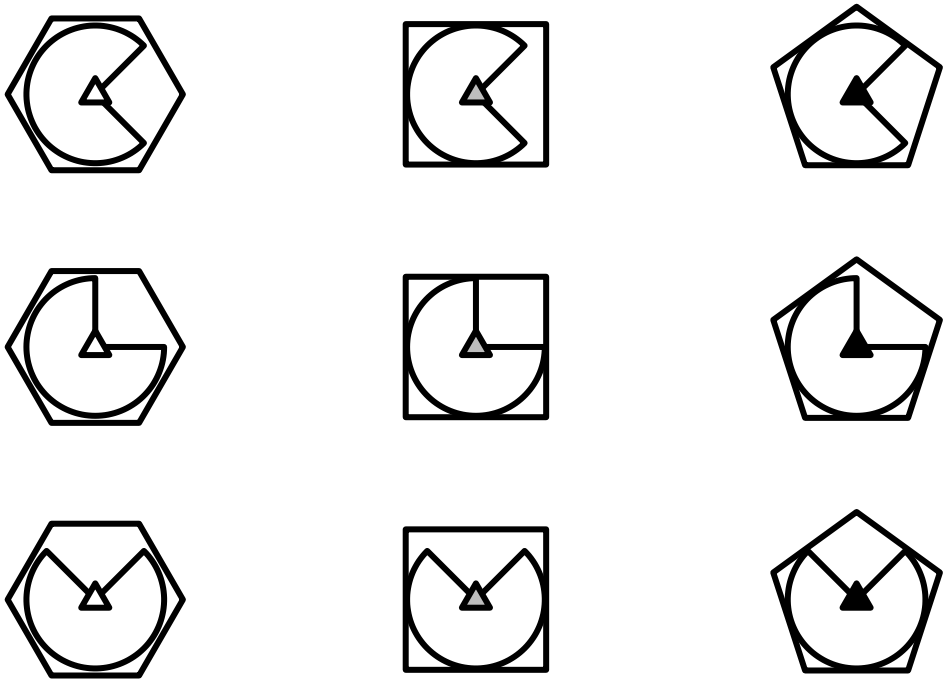
Figure 17: Single-layer matrix with two rules manipulated horizontally (Shape and filling) and one rules manipulated vertically (Orientation)

one direction and applied direct in the other, it will result in a TL-LR directional logic. On the other hand if the same rule is direct (or reverse) in both horizontal and vertical directional logic they will give rise to a LL-TR logic.

Concatenation of matrices

Matrices can be created either by applying the rule (or set of rules) with the `mat_apply()` function (i.e., single-layer matrix) or by concatenating different matrices (i.e., multi-layer matrix). Matrices can be concatenated together with the `com()` function (i.e., concatenation of matrices).

Figure 17 and Figure **??** depict a matrix created by the application of two horizontal rules (i.e., shade and shape) and one vertical rule (i.e., 'rotate"). As such, the figures in the changes and their shading change horizontally, while the rotation of the figures change vertically.



However, in Figure 17 the rules are all applied to the same set of figures (i.e., `hexagon()`, `square()`, `pentagon()`) to create one single matrix, while in Figure **??** the rules are separately applied to different figures to create the layers that are then combined to create the multi-layer matrix.

Each matrix concatenated to create the multi-matrix is a layer. The layering of the matrices moves from the background (considered as external) to the foreground (considered as internal). The most external layer is the background layer while the most internal layer is the foreground layer. Layers are counted inwards $m = 1, \ldots, M$ (where M is the total number of matrices to concatenate), such that the most external matrix (i.e., background matrix) is layer 1 and the most internal matrix (i.e., foreground matrix) is layer M.
The following illustrates the three matrices ($M = 3$) that have been concatenated to create the matrix in Figure **??**.

The matrix in Figure 18 is the first layer (i.e., background layer) and it is created via the horizontal application of the changing shapes rule `shape`.

Figure 19 is the second layer and it is created via the vertical application of the changing rotation rule `rotate`.

Finally, the matrix in Figure 20 is the first layer (i.e., background layer) and it is created via the horizontal application of the changing filling rule `shade`.

To create the multi-layer matrix in Figure **??**fig:multi-matrix), the matrices in Figures 18, 19, and 20 can be concatenated with the `com()` function, as follows:

```
com(multi_a, multi_b, multi_c)
```

The matrices have to be concatenated hierarchically by following the inward order of the layers.

The hierarchy between the layers is of the uttermost importance for definition of the distractors, specifically for the generation of the incomplete correlate ones. In such cases, the manipulation of the correct response is on the most internal layer (i.e., M). Further details on the definition and on the generation of the incomplete correlate distractors of multi-layer matrices are given in section Response list.

Generation of the response list

The generation of the distractors composing the response list is constrained by the type of matrix (i.e., single-layer matrix vs. multi-layer matrix), the rule(s) manipulated for the matrix generation, and the

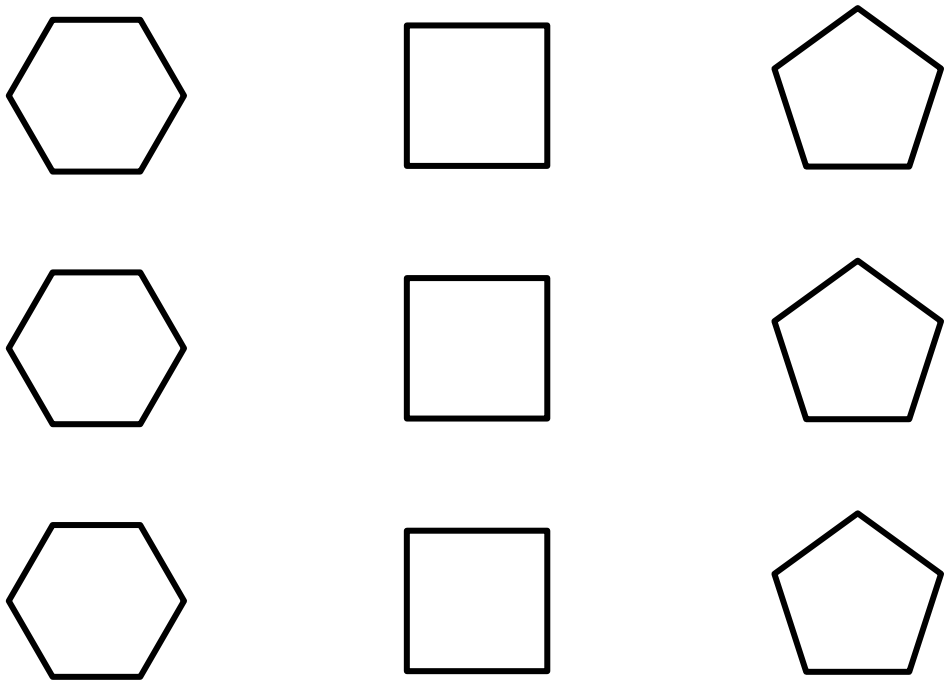


Figure 18: Layer 1 (Background matrix)

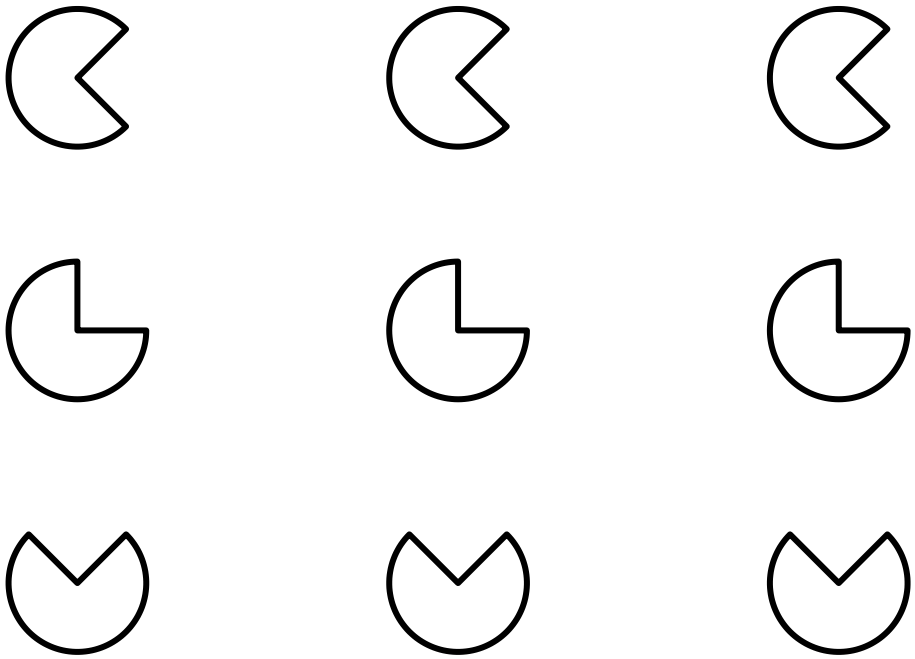


Figure 19: Layer 2

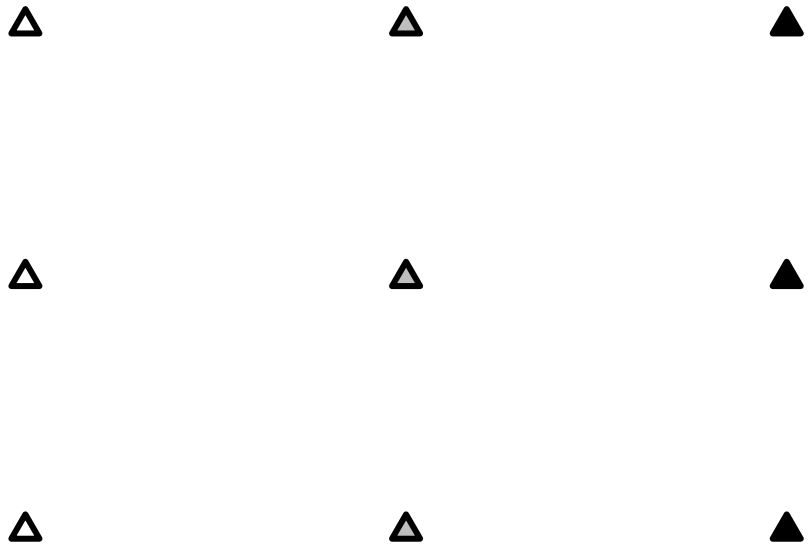


Figure 20: Layer 3 (Foreground matrix)

directional logic with which the rules are manipulated.
Given that, to the best of our knowledge, there is not a formal definition of the specific features of each distractor and on their applicability given the above-mentioned constrained, we started by giving a formal definition to each distractor and by considering all the possible exceptions given the constrained imposed by the matrix generated via the matrix operator.

The definitions of the distractors implemented in the **matRiks** package is reported in Table 4 .

The choice between cell SQ1 or cell SQ3 (WP-Copy, WP-Matrix, Difference distractors in 3×3 matrices) depends on the number of rules and on the directional logic with which they are manipulated. Specifically, cell SQ1 is selected when the matrix is generated via the manipulation of a single rule with V, H, or TL-LR directional logic. If the manipulation of the single rule follows a LL-TR directional logic, then the SQ3 cell is chosen. Cell Sq3 is chosen instead of cell Sq1 when 2 rules are manipulated with all possible directional logic (i.e., VV, HH, LL-TR, and TL-LR). This is done to avoid that the WP-Copy distractors could be interpreted as IC distractors. In other instances, the R distractors in 3×3 matrices cannot be generated because they are equal to the correct response. For instance, if a matrix is generated with the vertical manipulation of a single rule, the R-Left distractor (SQ8) is equal to the correct response.In such instances, the distractor that is equal to the correct response is generated by the response options operator but it is covered with a thick black cross.

A similar procedures is applied for choosing which cell to use between SQ3 and SQ2 or between SQ3 and SQ1 for the generation of WP-Matrix and Difference distractors in 2×2 matrices. In both cases, the choice depends on whether at least one rule is applied both horizontally and vertically. If the same rule is applied both horizontally and vertically, then SQ3 is chosen, otherwise SQ2 and SQ1.

The function for generating the response list (i.e., the correct response and all the distractor presented in the Table) is `response_list()`, which results in a list of figures that correspond to each of the distractors. For instance, the distractors of the matrix in Figure 17 can be obtained with:

```
response_list(single_matrix)
```

The function `response_list()` results in a named list of length 11 containing all the response options including the correct response:

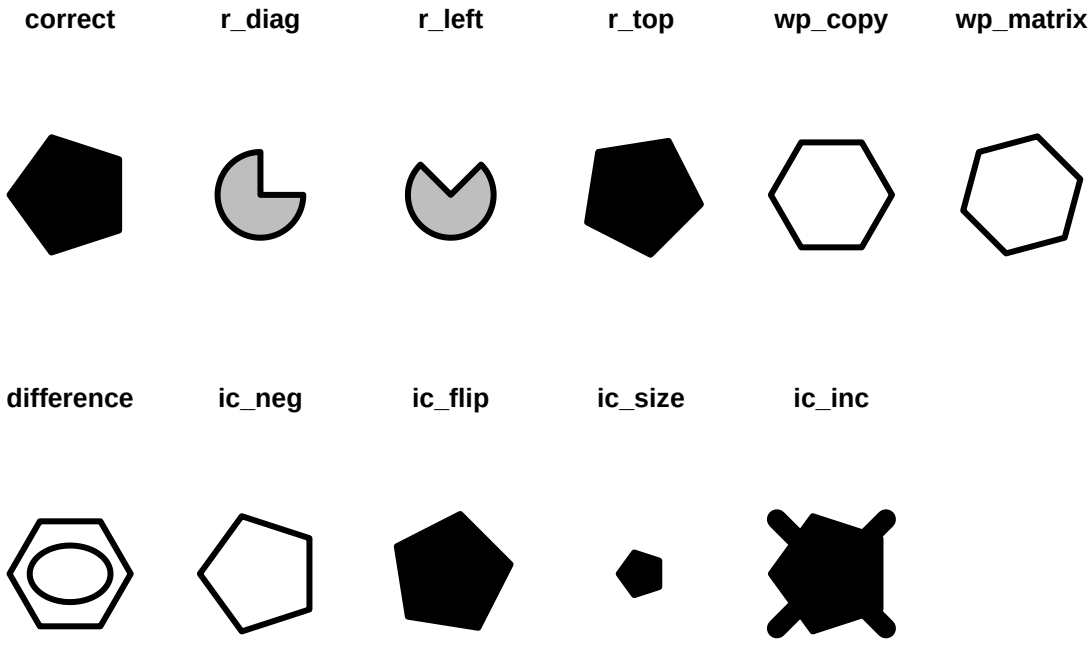
Table 4: Definition of the distractors implemented in the **matRiks** package for 3×3 and 2×2 matrices

Distractors	3×3 matrices
R-Left	SQ8
R-Top	SQ6
R-diag	SQ5
Wp-Copy	SQ1 or SQ3
WP-Matrix	SQ1 or SQ3 with the superimposition of another cell.
Difference	SQ1 or SQ3, SQ4, SQ7 with the superimposition of a figure which is not manipulated in the matrix
IC-Inc	It is the correct response with a missing element
	Single-Layer: Not possible
	Multi-layer: The most internal figure is removed from the correct response.
	Logic matrices: The element that is removed is randomly selected.
IC-Neg	Color inversion of the correct response (single-layer matrix) or of one of its figures (multi-layer ma
	Single-layer matrix: Color inversion of the figure in the correct response
	Multi-layer matrix: Color inversion of the most internal figure of the correct response
IC-Flip	Rotation or reflection of the correct response (single-layer matrix) or of one of its figures (multi-lay
	Single-layer matrix: Reflection/Rotation of the figure in the correct response
	Multi-layer matrix: Reflection/Rotation of the most internal figure of the correct response
IC-Scale	Resize of the correct response (single-layer matrix) or of one of its figures (multi-layer matrix)
	Single-layer: Resize of the figure in the correct response
	Multi-layer matrix: Only the most internal figure in the correct response is resized

```
#>  [1] "correct"    "r_diag"     "r_left"     "r_top"      "wp_copy"
#>  [6] "wp_matrix"  "difference" "ic_neg"     "ic_flip"    "ic_size"
#> [11] "ic_inc"
```

and it can be plotted as:

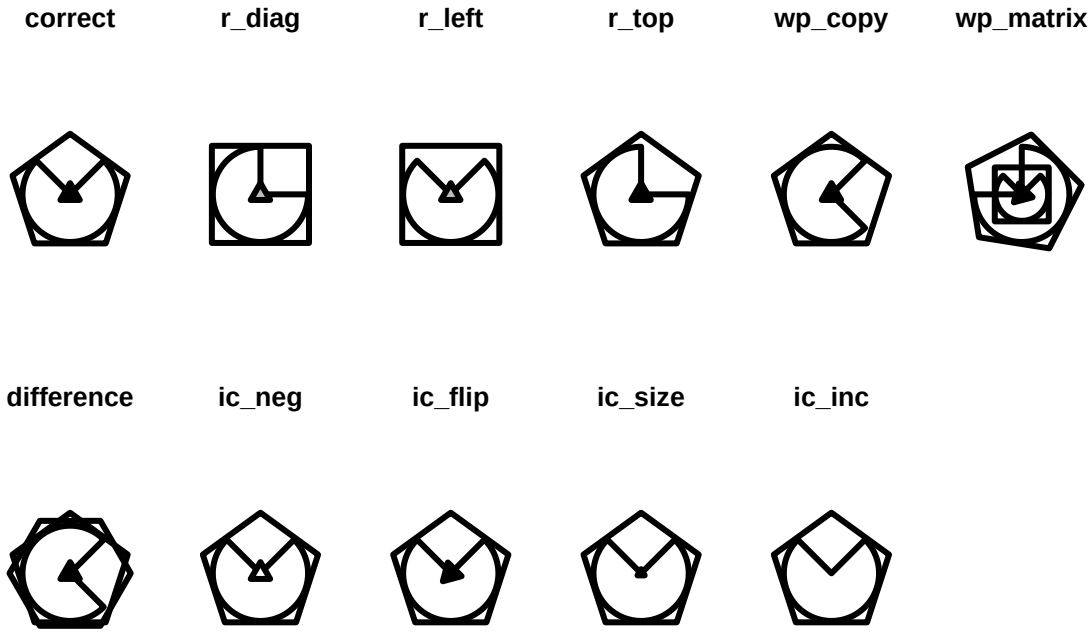
```
#> Warning in ic_inc.matriks(obj): IC-Inc cannot be obtained with a single figure
```



As it can be noted, a warning has appeared and it refers to the IC-Inc distractor. As per Table 4 .,

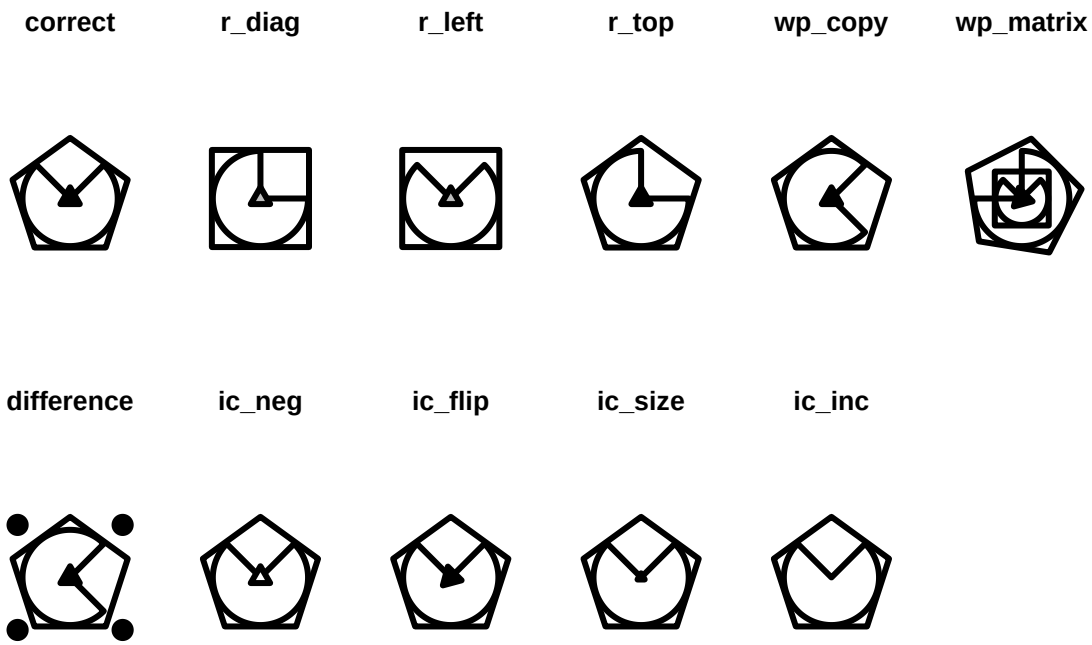
the distractor IC-Inc is defined as the correct response with a missing element. However, having a single-layer matrix does not allow for the removal of any element, hence the warning "IC-Inc cannot be obtained with a single figure" is thrown and the IC-Inc distractor is replaced by the correct response over which a black thick cross is imposed.

Given that the matrix in Figure **??** is composed of three layers, it is possible to obtain also the IC-Inc distractor:



However, the difference distractor is not well defined. The user can change the random seed for the generation of this distractor with the argument `seed()`, such that another random figure is chosen among the available ones:

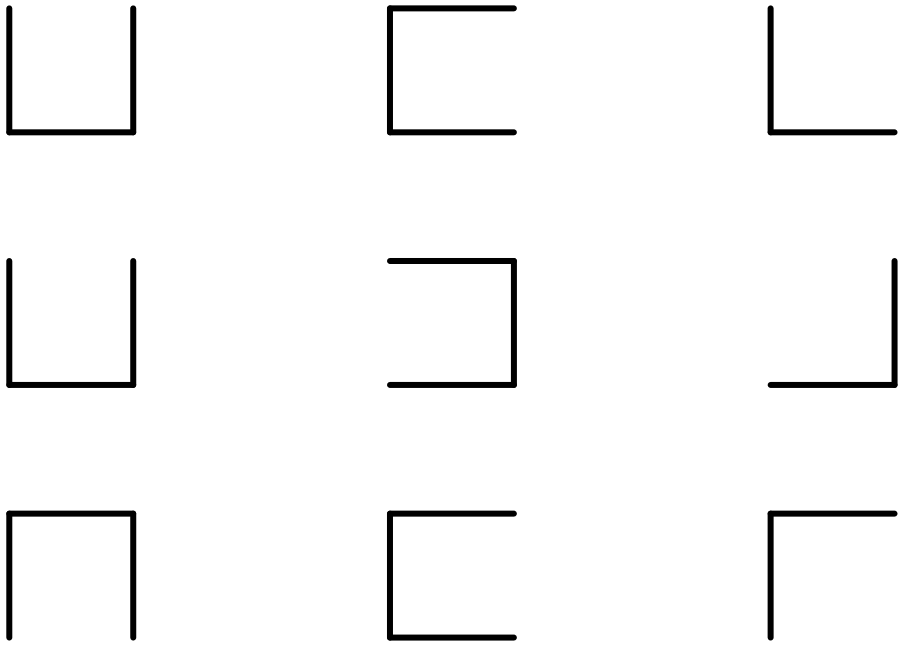
```
draw(response_list(multi_matrix, seed = 7),
     main = TRUE)
```



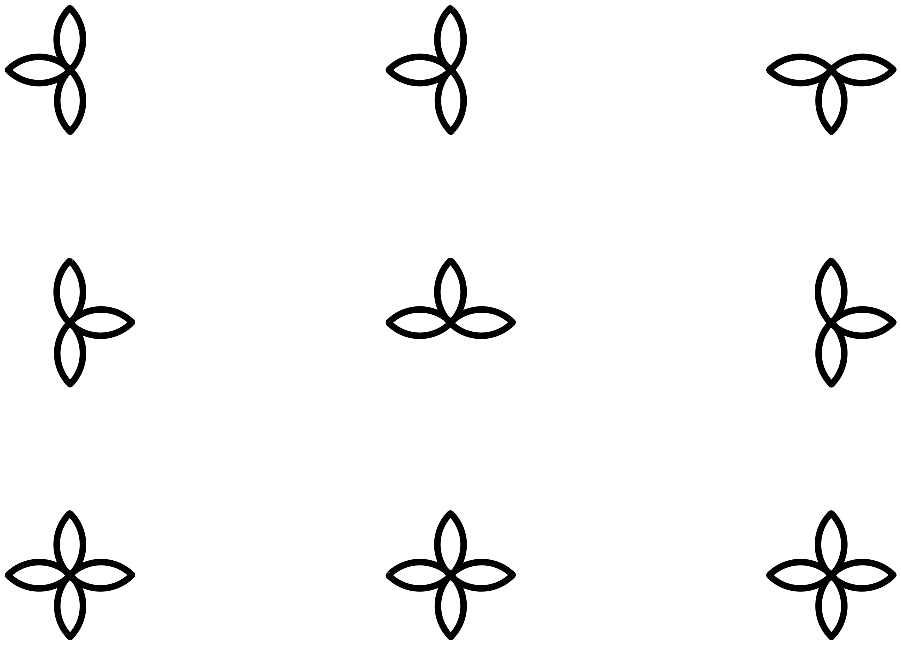
A complete example

This section presents a complete example on how to generate a multi-layer 3×3 matrix with logical rules and its related response options.

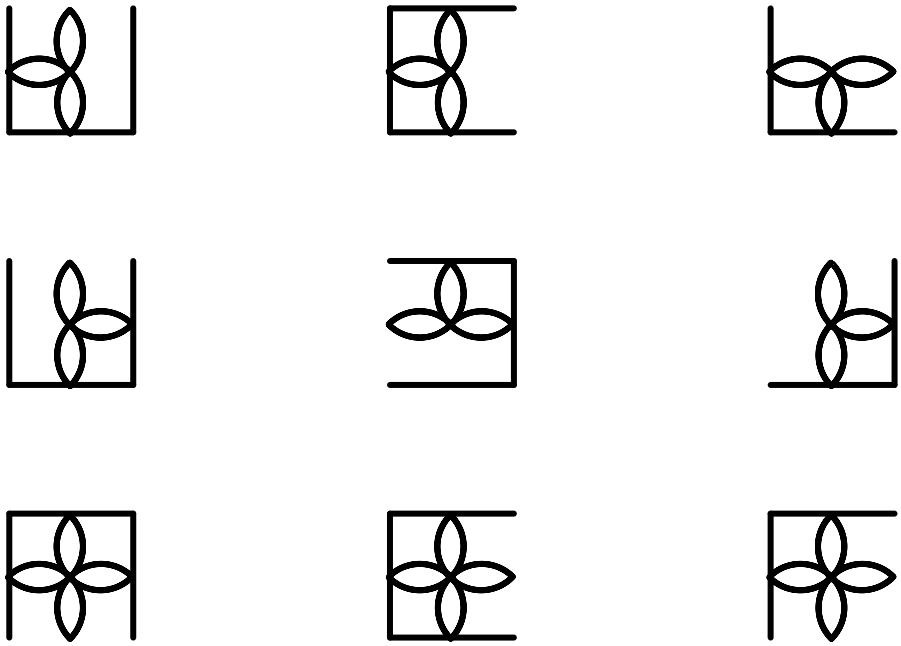
The first layer is generated by manipulating the AND logical rule according to an H directional logic on a square composed of 4 lines:



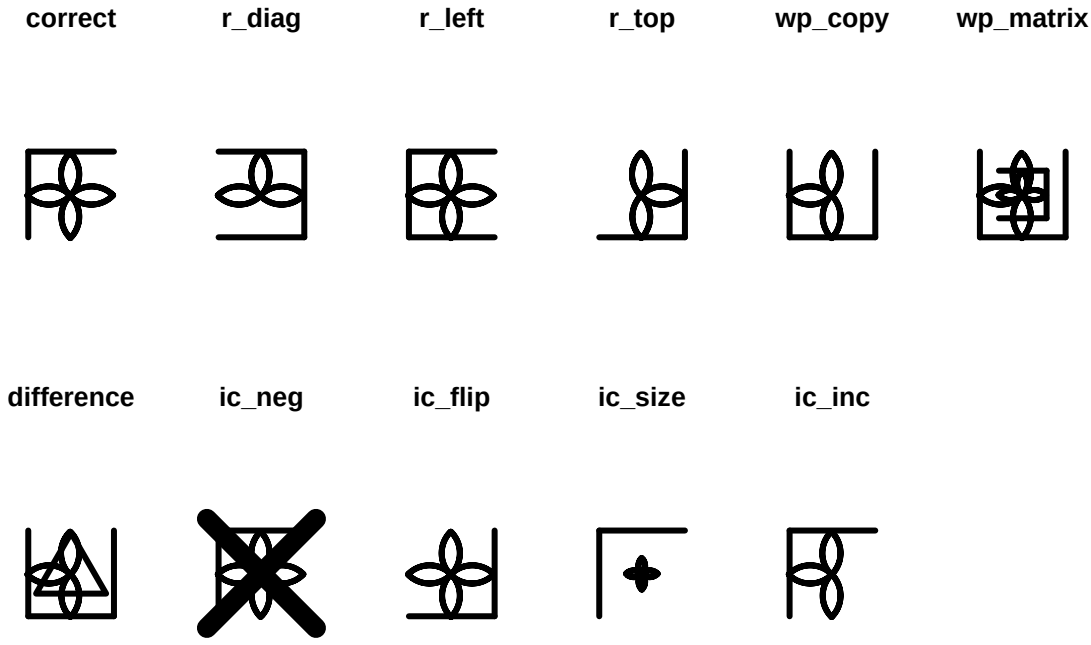
The second layer is generated by manipulating the OR logical rule according to a V directional logic on a flower composed of 4 petals:



The multi-layer matrix can be composed with the `com()` function by concatenating the two single-layer matrices:

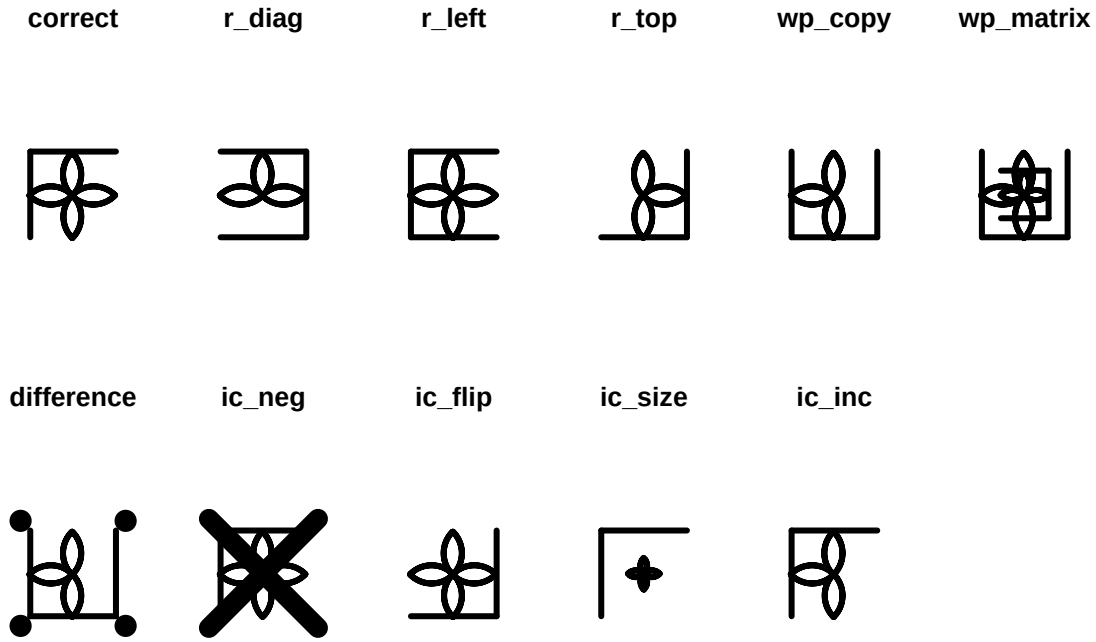


The response options associated with the multi-layer matrix can be generated with the `response_list()` function:

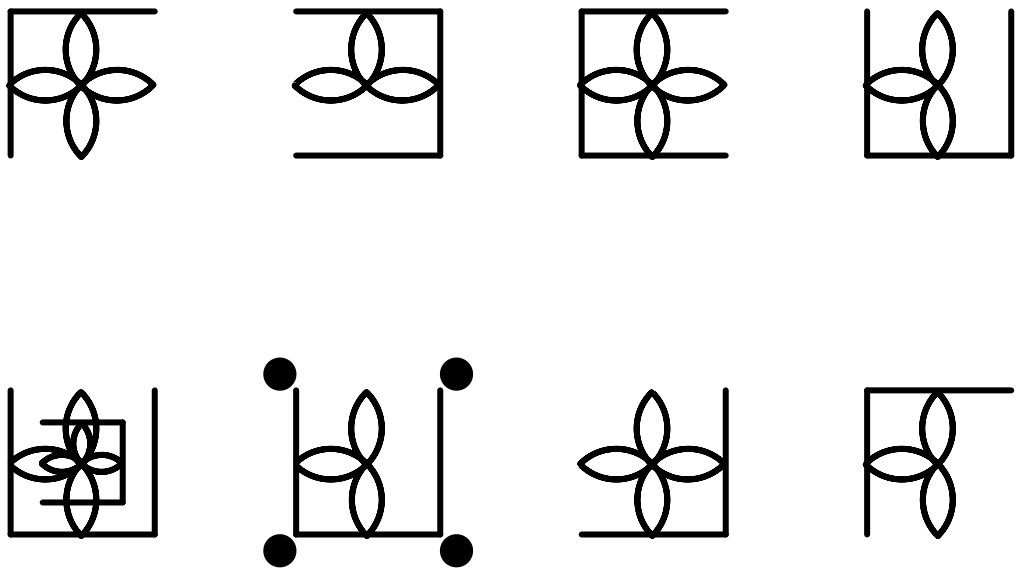


Since the filling color the `miley` cannot be changed, the IC-Neg distractor cannot be generated and the function throws a warning and the `ic_neg` distractor is replaced by the correct response with

the superimposition of a thick black cross. The `difference` distractor does not look good. The figure that is superimposed to the cell taken from the matrix can be changed by changing the seed in the `response_list()` function:



Assuming that a response list of length 8 (the correct response along with seven distractors) is associated with the multi-layer matrix, a character vector with the labels of the chosen distractors can be specified directly in the draw function:



4 Summary

This article briefly illustrates the functioning of the **matRiks** package for the automatic generation of Raven-like stimuli. This package has been developed with the intention of providing the users with a flexible, open-source, and easy-to-use tool for generating Raven-like matrices according to different rules, encompassing both visuo-spatial and logical rules, along with their associated response list.

The rules for generating the matrices and for generating the distractors associated to each matrix that are implemented in the **matRiks** package derive from vast literature concerning Raven's matrices and the error patterns observed on the CPM and the APM. As such, this package potentially provide the possibility for generating stimuli that are equivalent in terms of rules and response options to the standard Raven's stimuli. This allows for a comparison between the responses observed with the standard stimuli and those generated with the package.

This package has been developed within a broader project founded by the Italian Ministry of University and Research, which is aimed at the development of an intelligent system for the adaptive assessment of executive functions and fluid intelligence among general and clinical populations. The test used for assessment of fluid intelligence (named MatriKS) was developed following the principles of Raven's Matrices. Specifically, the stimuli composing the test have been developed with the **matRiks** package ((**mdpi?**)).

This package allows for high degrees of freedom in the generation of matrices with different difficulty. For instance, it allows for the generation of stimuli that combine multiple rules by exploiting the possibility of layering multiple matrices together. Moreover, the high degree of control provided to the users allow them to thoroughly manipulate the difficulty of the matrix, for instance by adding or removing objects and combining together different rules.

The package is regularly maintained and new functions will be available in the future. As such, the users should refer to the official documentation of the package that is constantly updated.

Although the functions implemented in **matRiks** are quite straightforward and easy to use, they need a basic knowledge of the R language. To overcome this issue and allow for a wider use of the package even among people that are not familiar with R, a web application built with the **shiny** package ((**shiny?**)) will be developed in the future.

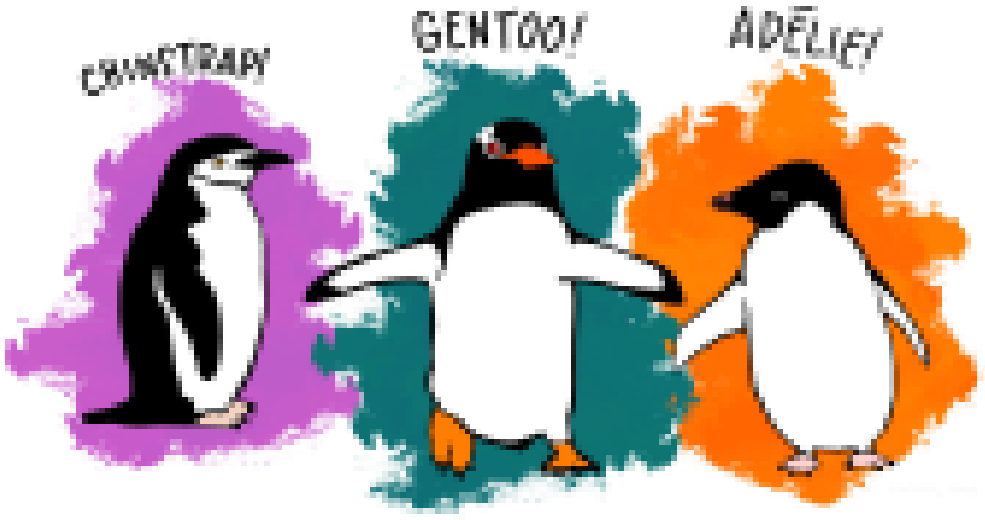


Figure 21: Artwork by allison_horst

Table 5: A basic table

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
Adelie	Torgersen	39.1	18.7	181	3750	male	2007
Adelie	Torgersen	39.5	17.4	186	3800	female	2007
Adelie	Torgersen	40.3	18.0	195	3250	female	2007
Adelie	Torgersen	NA	NA	NA	NA	NA	2007
Adelie	Torgersen	36.7	19.3	193	3450	female	2007
Adelie	Torgersen	39.3	20.6	190	3650	male	2007

5 Customizing tooltip design with ToOoOlTiPs

ToOoOlTiPs is a packages for customizing tooltips in interactive graphics, it features these possibilities.

6 A gallery of tooltips examples

The **palmerpenguins** data (Horst, Hill, and Gorman 2020) features three penguin species which has a lovely illustration by Alison Horst in Figure 21.

Table 5 prints at the first few rows of the penguins data:

Figure 22 shows an plot of the penguins data, made using the **ggplot2** package.

```
penguins %>%
  ggplot(aes(x = bill_depth_mm, y = bill_length_mm,
             color = species)) +
  geom_point()
```

7 Summary

We have displayed various tooltips that are available in the package **ToOoOlTiPs**.

References

Brancaccio, Andrea, Ottavia M. Epifania, and Debora de Chiusole. 2023. *matRiks: Generates Raven-Like Matrices According to Rules*. https://CRAN.R-project.org/package=matRiks.

Cattell, Raymond B. 1963. "Theory of Fluid and Crystallized Intelligence: A Critical Experiment." *Journal of Educational Psychology* 54 (1): 1. https://doi.org/https://doi.org/10.1037/h0046743.

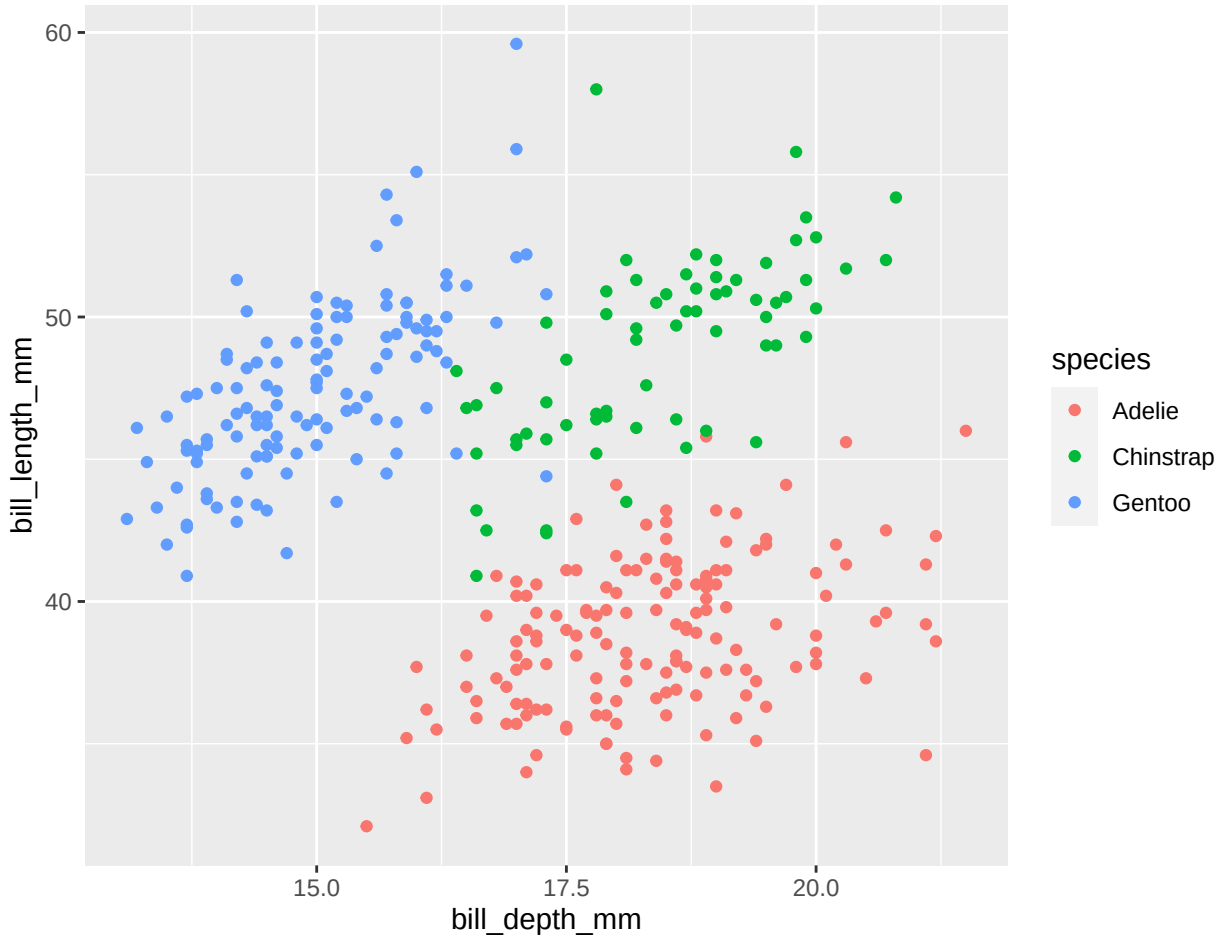


Figure 22: A basic non-interactive plot made with the ggplot2 package on palmer penguin data. Three species of penguins are plotted with bill depth on the x-axis and bill length on the y-axis. Visit the online article to access the interactive version made with the plotly package.

Forthmann, Boris, Natalie Förster, Birgit Schütze, Karin Hebbecker, Janis Flessner, Martin T. Peters, and Elmar Souvignier. 2020. "How Much g Is in the Distractor? Re-Thinking Item-Analysis of Multiple-Choice Items." *Journal of Intelligence* 8 (1): 11. https://doi.org/10.3390/jintelligence8010011.

Horst, Allison Marie, Alison Presmanes Hill, and Kristen B Gorman. 2020. *palmerpenguins: Palmer Archipelago (Antarctica) Penguin Data*. https://allisonhorst.github.io/palmerpenguins/.

Kunda, Maithilee, Isabelle Soulières, Agata Rozga, and Ashok K. Goel. 2016. "Error Patterns on the Raven's Standard Progressive Matrices Test." *Intelligence* 59 (November): 181–98. https://doi.org/10.1016/j.intell.2016.09.004.

Raven, JC ea. 1938. "Raven's Progressive Matrices." *Western Psychological Services* 2: 5.

Raven, John, and H. Raven. 2004. "Manual for Raven's Progressive Matrices and Vocabulary Scales, Section 3: The Standard Progressive Matrices, Including the Parallel and Plus Versions. 2000 Edition, Updated 2004." In, Section 3.

Storme, Martin, Nils Myszkowski, Simon Baron, and David Bernard. 2019. "Same Test, Better Scores: Boosting the Reliability of Short Online Intelligence Recruitment Tests with Nested Logit Item Response Theory Models." *Journal of Intelligence* 7 (3): 17. https://doi.org/10.3390/jintelligence7030017.

Quietest Quokka
University of Little Mates
Department of Letter Q
Somewhere, Australia
https://www.britannica.com/animal/quokka
ORCiD: 0000-1721-1511-1101
qquo@ulm.edu

Bounciest Bilby
University of Little MatesUniversity of Aussie Animals
Department of Letter Q, Somewhere, Australia
Department of Marsupials, Somewhere, Australia
https://www.britannica.com/animal/bilby
ORCiD: 0000-0002-0912-0225
bbil@ulm.edu